VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

# OPTIMALIZACE ULOŽENÍ MATERIÁLU PRO PŘEPRAVU

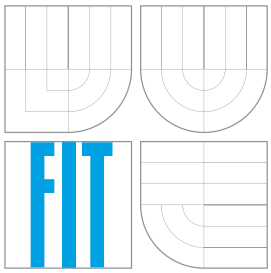
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL VACEK

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

# OPTIMALIZACE ULOŽENÍ MATERIÁLU PRO PŘEPRAVU

STUFF PLACING OPTIMIZATION FOR TRANSPORT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MICHAL VACEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. BOHUSLAV KŘENA, Ph.D.
SUPERVISOR

BRNO 2007

# Abstrakt

Práce se zabývá aplikací pro optimalizaci uložení materiálu pro přepravu. Aplikace je vyvíjena ve spolupráci se společností Škoda Auto a.s., která řeší problém přepravy materiálu v kontejnerech z výrobních hal v České republice do montážních závodů v zahraničí. Zde je popsána její analýza, návrh i implementace. Aplikace se skládá ze dvou částí. První vypočítá optimalizaci – vybere z množiny modulů (krabic) vhodnou kombinaci a umístí ji kontejneru. Druhá část aplikace toto rozložení zobrazuje. Část této práce je také věnována příbuzným optimalizačním metodám, zejména pak Knapsack a Bin-packing problému.



# Klíčová slova

Optimalizace, uložení materiálu, Knapsack, Bin-packing problem



# Abstract

This bachelor thesis treats of application of stuff placing optimization for transport. Application is developed in cooperation with Škoda Auto a.s., which solves the problem of transport materials in containers from their factories to assembly halls abroad. It describes an application analysis, a plan and an implementation. Application consist of two parts. The first computes the optimalization – it selects subset from set of moduls (boxes) and it locates this boxes to a container. The second part of application display the stuff placing. The part of this bachelor thesis is also set of agnated optimalization methods, mainly Knapsack and Bin-packing problem.



# Keywords

Optimalization, stuff placing, Knapsack, Bin-packing problem



# Citace

Michal Vacek: Optimalizace uložení materiálu pro přepravu, bakalářská práce, Brno, FIT VUT v Brně, 2007

Optimalizace uložení materiálu pro přepravu

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Bohuslava Křeny. Další informace mi poskytli zaměstnanci společnosti Škoda Auto a. s. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Vacek
10. května 2007

## Poděkování

Rád bych poděkoval Ing. Bohuslavu Křenovi, Ph.D., za pomoc s přípravou bakalářské práce a také všem zaměstnancům společnosti Škoda Auto a. s., kteří se mnou na vývoji aplikace spolupracovali.

# Obsah

# Kapitola 1

# Úvod

Cílem práce je popsat aplikaci, kterou jsem vytvořil ve spolupráci se společností Škoda Auto a.s. (dále pouze Škoda). V této společnosti vykonávám externí stáž a dostal jsem možnost propojit vývoj optimalizační aplikace s mojí bakalářskou prací. V tomto dokumentu je tedy popsán návrh a implementace všech částí programu včetně ovládání a komunikace s vnitřními systémy Škody.

Společnost řeší problém přepravy materiálu z výrobních hal v České republice do montážních závodů v zahraničí. Z ekonomických důvodů, především kvůli vysokým clům na hotové výrobky v některých zemích, se nevyplatí do daných států vyvážet celá auta. V různém stupni rozložení se tak jednotlivé součástky balí do krabic nebo palet (označovaných dále jako *moduly*) a poté se posílají do zahraničí v kontejnerech či vagónech.

Přeprava materiálu je dosti nákladná a promítá se značně v ceně všech výrobků, problém vhodně využít nákladový prostor proto trápí nejen expediční firmy po celém světě.

Úkolem tak bylo optimalizovat naložení kontejneru (resp. vagónu), neboli vypočítat způsob, jak jednotlivé moduly v kontejneru nejlépe uspořádat. A výsledek poté vhodně zobrazit. Detaily zadání jsem shrnul v kapitole 2.

## 1.1 Optimalizace

Nejdříve se podívejme, co *optimalizace* vůbec znamená. Jde vlastně o matematickou disciplínu, která hledá minimum (resp. maximum) dané funkce $f(x)$ na množině přípustných řešení $M$. Někdy, stejně jako v našem případě, pomáhají optimalizační úlohu řešit tzv. *podmínky optimality*. Jde o podmínky, které musí platit pro optimální řešení, a slouží především k redukci množiny přípustných řešení. [4] Takovéto podmínky naší optimalizace jsou popsány v podkapitole 2.2. Podmínky lze rozdělit na:

- nutné (musí splňovat každé optimální řešení úlohy)
- postačující (pokud je nějaký bod splňuje, automaticky jde o optimální řešení)

Optimalizaci a především optimalizačním metodám příbuzným našemu problému se věnuje kapitola 3.

## 1.2 Struktura bakalářské práce

V následujících kapitolách jsem popsal vývoj aplikace (od návrhu až po implementaci a testování), která má zefektivnit cyklus balení a nakládky materiálu a především snížit

náklady na jeho přepravu. Jelikož aplikace ještě nedostala své "jméno", používám dočasně pracovní označení *b2c*, což jsou počáteční písmena anglického spojení *boxes to containers*. Jednotlivé kapitoly obsahem víceméně kopírují body zadání bakalářské práce.

V první kapitole 2 detailně specifikuji zadání. Popíšu zde problémy, které musí aplikace řešit, a požadavky, které jsou na program kladeny ze strany Škody. Ve druhé kapitole trochu odbočím. Projdu příbuzné metody, které se rovněž zabývají problémem kombinační optimalizace, zejména Knapsack a bin-packing problem. Tím vytvořím dostatečnou teoretickou půdu pro vytvoření vlastního optimalizačního algoritmu.

Další dvě kapitoly se zabývají návrhem (viz. kapitola 4), resp. implementací (viz. kapitola 5) aplikace. Tvoří hlavní část bakalářské práce a jsou v nich popsány všechny části aplikace, způsob, jakým byly vytvořeny, a funkce, které zastávají. Popíšu zde také princip, jakým je optimalizace vypočítávána a zahrnuta bude také podkapitola věnující se testování a měření vlastností aplikace.

V poslední kapitole shrnu dosažené výsledky. Upřesním, do jaké míry se podařilo splnit požadavky na aplikaci a jakým způsobem je možné program dále rozšiřovat a vyvíjet.

Doplňující tabulky a grafy s výrazně popisným charakterem jsem umístil do příloh. Nalezneme zde i uživatelský manuál, který jsem k aplikaci vytvořil.

# Kapitola 2

# Specifikace zadání

V této kapitole budou analyzovány všechny detaily zadání, nároky, které musí aplikace splňovat, a nastíněna bude i struktura celé aplikace. Specifika zadání jsem průběžně konzultoval s odpovědnými lidmi ve Škodě tak, aby program byl následně použitelný v praxi, a to konkrétně v expedičních halách V8 v Mladé Boleslavi.

Hlavním požadavkem Škody bylo rozdělit aplikaci na dvě nezávislé části. Jeden program má optimální rozložení modulů v kontejneru vypočítat a druhý ho vhodně zobrazit. Důvodem je možnost samostatné obsluhy obou částí. První (optimalizační) program budou obsluhovat lidé, kteří vytváří balící plán pro expedici. Zatímco druhý (zobrazovací) program budou používat zejména lidé, kteří materiál balí či ho přímo nakládají do kontejnerů.

Z tohoto rozdělení vyplývá i první otázka: jak spolu oba dva programy mají komunikovat?! Zvolen byl ten nejjednodušší způsob. Optimalizační program výsledky uloží do souboru a zobrazovací program jej odtud bude číst. Tato varianta byla vybrána i s ohledem na hlavní požadavek Škody, aby celá aplikace jakýmkoliv způsobem nezasahovala do vnitřních systémů společnosti, a veškerá komunikace tak probíhala pouze na úrovni *čtení/zápisu* z/do souborů.

## 2.1 Vstupní parametry pro optimalizaci

Seznam materiálu, který má být expedován, je znám přibližně na 5 dní dopředu. Ke každému materiálu přísluší modul, do kterého se materiál balí. Seznam nejpoužívanějších modulů i s jejich rozměry je uveden v tabulce A.3. Důležité je rozdělit materiál podle dne, kdy má být expedován, a hlavně závodu, kam má být poslán. Všechna data jsou pak uložena ve vstupních souborech, strukturou souborů se zabývá sekce 4.1.1.

Důležité je také zjistit, do jak velkého kontejneru (resp. vagónu) má být materiál naložen, tedy pro jak velký prostor se má optimalizace vypočítat. Rozměry standartních kontejnerů či vagónů jsou známy, ale pro různé nečekané situace či pro testování a další vývoj by měla být možnost tyto rozměry zadat i ručně.

Dále mezi požadavky byla doplněna možnost zadat míru zaplnění. V praxi není téměř možné dosáhnout 100 % zaplnění kontejneru, tato míra zaplnění proto udává, jaké zaplnění je dostatečné pro ukončení optimalizačního výpočtu (touto aproximací algoritmu se zabývá podkapitola 4.3).

## 2.2 Omezení a kritéria při výpočtu optimalizace

Při skládání jednotlivých modulů do kontejneru narážíme na řadu problémů. V této podkapitole je uveden jejich stručný přehled, který do značné míry ovlivňuje samotný algoritmus výpočtu.

### 2.2.1 Stohovatelnost

Zřejmě nejvíce omezujícím faktorem je hmotnost materiálu uloženého v modulu a také konstrukce onoho modulu. Nesmí totiž dojít k přetížení či narušení modulů umístěných vespod kontejneru, aby nedošlo ke znehodnocení materiálu.

Při skládání jednotlivých modulů na sebe existuje jednoduché pravidlo. Na menší bednu nesmí přijít větší z důvodů jejich nosnosti. I zde existují vyjímky – systém modulů GLT. Jedná se o vysokonosné bedny, které svými rozměry do sebe "zapadají". Tyto moduly tvoří kostru celého kontejneru, a jako jediné je lze stavět na sebe způsobem, kdy pod velkou bednu přijdou 2 až 4 malé. Přehled pravidel, jak lze takto moduly GLT stohovat je uveden v tabulce A.1.

### 2.2.2 Vkládání malých modulů do velkých

Jedním z řešení malé nosnosti ostatních modulů je jejich vkládání do GLT. V praxi se vkládají některé menší moduly KLT a MOD primárně do GLT5754. Pouze pokud není do tohoto modulu dostatečný počet kusů materiálu, vkládají se do GLT5755. Po kolika kusech se tyto moduly do GLT modulů vkládají je uvedeno v tabulce A.2. Zbylé moduly, kterých není dostatečný počet, se používají k vyplnění vrchních míst v kontejneru.

### 2.2.3 Ostatní omezení

Dalším zajímavým problémem byla možnost rotace modulů, ty se totiž mohou otáčet pouze podle $y$ osy, tedy nikoliv převracet. Důvod je zřejmý, nesmí dojít k poškození materiálu uvnitř.

Mezi poslední požadavek Škody patří možnost bezproblémového naložení daných modulů do kontejneru. Dveře kontejneru nesahají přes jeho celou čelní stranu, ale odzhora chybí přibližně 20 cm. Tudíž nelze v posledních dvou metrech kontejneru skládat moduly až ke stropu, vysokozdvižný vozík by se tam nedostal. Tento problém odpadá u vagonů, které se nakládají ze strany.

## 2.3 Požadavky na zobrazení výsledku

Při zobrazení výsledku, tedy rozložení modulů v kontejneru, je kladen důraz především na přehlednost. Program by měl jednoduše zobrazit jednotlivé moduly a postupně znázornit jejich umístění do kontejneru. Také musí zobrazit veškeré dostupné informace o daném modulu a materiálu v něm, aby byla možná jejich snadná identifikace. Program by si měl pamatovat všechny dosud nevyexpedované kontejnery, které se identifikují pomocí ID čísla (přiděleno během optimalizačního výpočtu).

# Kapitola 3

# Příbuzné optimalizační metody

Nyní trochu odbočíme od našeho tématu. Podíváme se detailněji na různé problémy, které mají určité podobné vlastnosti jako naše optimalizace. Algoritmy řešící problém kombinační optimalizace jsou charakteristické tím, že hledají maximální ze všech nabízených možností limitované nějakým omezením. Mezi nejznámější patří *Knapsack* nebo *Bin-packing problem*. Pro mnohé optimalizační problémy je však obtížné navrhnout algoritmy, které je vyřeší optimálně a zároveň rychle (např. pro NP-úplné problémy). V takovém případě studujeme tzv. aproximační algoritmy, které pracují rychle, a najdou řešení více či méně blízké optimálnímu řešení. Takovéto aproximace využívá i algoritmus, který jsem pro výpočet použil (viz. podkapitola 4.3)

## 3.1 Knapsack

Mezi nejzajímavější příklady kombinačních algoritmů patří *Knapsack problem*, v češtině též označován jako *problém batohu*. Jedná se o libovolně přesně aproximovatelný problém. Zjednodušeně řečeno vybíráme z kolekce položek, z nichž každé má svoji váhu a hodnotu, jejich nejlepší kombinaci, která se vejde do vymezeného prostoru (již zmiňované omezení), a přitom hodnota složek bude co možná největší.

Mezi základní verze patří tzv. `0/1 knapsack problem`. Zde počet jednotlivých položek může nabývat pouze hodnot nula nebo jedna, tedy buď je v řešení položka zahrnuta nebo není. Pokud omezíme počet položek nějakým číslem, mluvíme o `bounded knapsack problem`.

### 3.1.1 Formální definice 0/1 knapsack problému

Mějme $n$ položek a $m$ úložných prostorů,

- $p_{i,j}$ hodnota položky $j$ v prostoru $i$
- $w_{i,j}$ váha položky $j$ v prostoru $i$
- $c_i$ kapacita prostoru $i$.

Hledáme vektor

$$x = x_1, x_2, \ldots, x_n \in \{0,1\}^n, kde\ x_j = 1,\ pokud\ je\ prvek\ zvolen, \tag{3.1}$$

$$f(x) = (f_1(x), f_2(x), \ldots, f_m(x))\ je\ maximum,\ kde \tag{3.2}$$

$$f_i(x) = \sum_{j=1}^{n} p_{i,j} * x_j,\ a\ kde \tag{3.3}$$

$$\vee i \in 1, 2, \ldots, m : \sum_{j=1}^{n} w_{i,j} * x_j \leq c_i. \tag{3.4}$$

Složitost řešení knapsack problému poté záleží na kapacitě jednotlivých úložných prostorů. Pro testování a porovnávání rychlostí jednotlivých algoritmů se používá obecně hodnota $c_i = 0.5 \sum_{j=1}^{n} w_{i,j}$. [2]

Řešení může probíhat mnoha metodami. Od "hrubé síly", kdy se neustále zjišťuje lepší výběr položek. Po řešení heuristikou podle poměru hodnota/váha. Ta je díky menší asymptotické složitosti použitelná i pro instance s větším počtem prvků. Danou heuristiku můžeme ještě rozšířit o tzv. *testování nejcennější věci.*

## 3.2 Bin-packing problem

Pomocí *bin-packing problému* zkoušíme minimalizovat počet úložných prostorů (tzv. *bins*) potřebných k naložení určitého počtu položek. Otázku lze také ovšem pojmout z opačného pohledu, pokusit se do jednoho úložného prostoru naskládat co nejvíce položek, resp. co nejlépe vyplnit daný prostor.

Právě tato varianta má mnoho společného s naší optimalizací a tudíž si ji detailněji rozebereme. Mějme *bin* a množinu krabic, jejichž celkový objem je větší než objem úložného prostoru. Úkol je vybrat podmožinu daných krabic a rozmístit ji do prostoru tak, aby zabrala co nejvíce místa. Pokud $V$ je objem prostoru a $V_b$ je celkový objem všech krabic, využití každé krabice je potom $V_b/V$. V závislosti na počtu rozměrů krabic a prostoru poté hovoříme o *2D bin-packing* (pokud mají pouze šířku a výšku) nebo o *3D bin-packing* problému (pokud mají i hloubku). V prvním případě se jedná v podstatě o množinu obdélníků, ve druhém pak o množinu kvádrů. Zde úmyslně vynechávám možnosti, kdy se nejedná o symetrické tvary, jelikož zde je výpočet ještě daleko náročnější. [1]

Algoritmus výpočtu je založen na neustálém přidávání a odebírání krabic do úložného prostoru, často k tomu využívá rekurzi. Pokud bychom nechali proběhnout algoritmus celý, tzn. našel by všechny možnosti, jak položky do prostoru umístit, jednoduše bychom z nich vybrali tu nejlepší. V našem případě tu, která zabrala nejvíce místa. Ovšem jsou zde dva problémy. Kam umístit první položku, od které se celý výpočet provádí?! Pokud bychom zkoušeli umístit do daného prostoru všechny položky (což bychom museli, pokud chceme zjistit všechny možnosti rozložení), a přitom je zkoušeli postupně umístit do všech částí prostoru, již při poměrně malém počtu položek by byl výpočet neúnosně časově náročný. Zde přichází na řadu již zmiňovaná aproximace. Nejefektivnější místo, kde výpočet začít, je jeden z rohů úložného prostoru. Tím snížíme náročnost výpočtu.

Druhý problém ovšem může být, že pokud máme několik tisíc položek (jako například v našem případě), výpočet bude trvat i tak příliš dlouho. Druhou aproximací může být stanovení určité úrovně, kdy lze považovat řešení za dostatečné. Ovšem to už nemusí být nejlepší možné.

# Kapitola 4

# Návrh aplikace pro optimalizaci včetně grafické reprezentace

V této kapitole budou detailně popsány jednotlivé části aplikace, jejich funkce a vzájemná komunikace, zejména z pohledu uživatele. Popsáno bude chování i ovládání aplikace. Jak již bylo naznačeno v popisu požadavků, aplikace je rozdělena do dvou na sobě nezávislých částí.

S ohledem na obecnost aplikace a možnost jejího pozdějšího širšího vyžití byla ještě optimalizační část rozdělena na další dva celky. Kostru aplikace tak tvoří samotný optimalizační program `b2c.exe`, který provádí vlastní výpočet uložení materiálu v kontejneru. K tomu využívá všechny vstupní soubory, které jsou popsány v sekci 4.1.1 a výsledek ukládá do souboru `res.dat`. Nadstavbou tohoto programu je grafické uživatelské rozhraní, které slouží k jednoduchému získání parametrů od uživatele. Vypočtenou optimalizaci následně program `b2cII.jar` zobrazí. Celý optimalizační cyklus včetně zobrazení je znázorněn na obrázku 4.1.

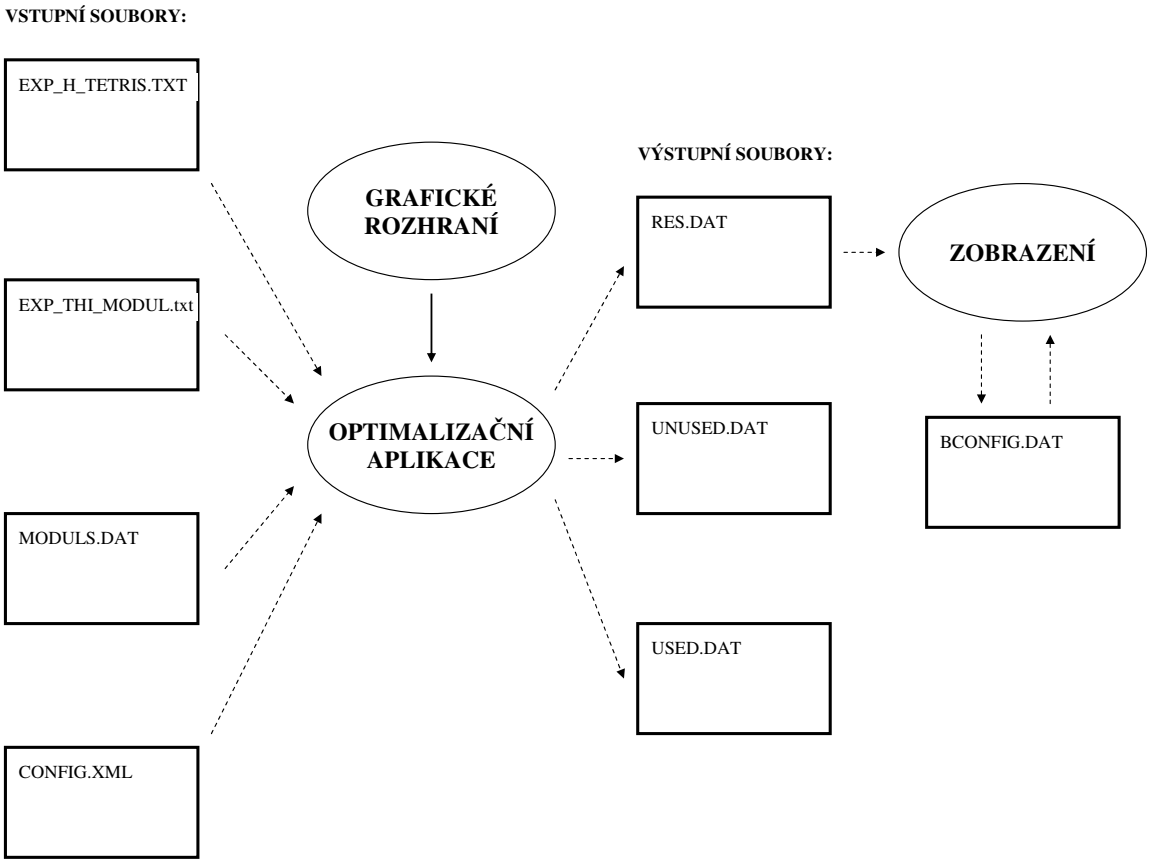
## 4.1 Struktura souborů

Veškerá komunikace jak mezi jednotlivými částmi, tak mezi samotnou aplikací a vnitřními systémy Škody, probíhá pomocí souborů. V této podkapitole budou detailně popsány struktury jednotlivých souborů, které jsou pro činnost aplikace nezbytné. Rozdělit bychom je mohli do tří kategorií. První tvoří vstupní soubory, které obsahují vstupní data pro optimalizaci. Druhou poté výstupní soubory, kde jsou uloženy výsledky, a poslední kategorii tvoří konfigurační soubory, které obsahují potřebné informace pro činnost jednotlivých částí aplikace.

### 4.1.1 Vstupní soubory

Nejdůležitější vstupní soubor je `EXP_H_TETRIS.txt`. Obsahuje již zmiňovaný seznam materiálu, který je v danou chvíli možné zahrnout do výpočtu rozložení. Příklad obsahu tohoto souboru je v tabulce A.4. Soubor obsahuje:

- kód závodu, kam je materiál expedován
- den, kdy má být materiál expedován, a kód směny
- kód materiálu




Obrázek 4.1: Struktura optimalizační cyklu

- počet kusů
- závěsku (tj. číslo krabice sloužící k identifikaci)
- hmotnost jednoho kusu materiálu

Po zpracování těchto informací, vybrání odpovídajících materiálů, je nutné jednotlivé kódy "rozluštit", abychom věděli, do jakých modulů je materiál balen – tedy například kolik zabírá místa. K tomu slouží další dva soubory `EXP_H_MODUL.txt` a `moduls.dat`. V prvním jmenovaném souboru je číslovaný popis postupu balení jednotlivých materiálů řazený vzestupně, znázorněno v tabulce A.5. Tento "oficiální" (nesnadno pochopitelný) popis zjednodušeně znamená, že v souboru lze nalést postup, jak s jednotlivými materiály zacházet, do čeho je postupně ukládat a například jaké bezpečnostní fólie použít. Na konci tohoto cyklu je pochopitelně uveden i modul, ve kterém je materiál převážen (a právě ten je důležitý). Jelikož je tento seznam řazen vzestupně (opačně než se balí), pro naši optimalizační aplikaci je potřebná pouze první položka. Rozměry a nosnost jednotlivých modulů je popsána v druhém souboru `moduls.dat` (viz. tabulka A.3).

Po zpracování souboru `EXP_H_TETRIS.txt`, a dohledání všech potřebných informací v dalších dvou souborech, získáme úplný seznam modulů a materiálů v nich, ze kterých

posléze počítáme optimální rozložení v nákladovém prostoru. Jakým způsobem se z jednotlivých souborů čtou údaje a v jakém pořadí se zabývá podkapitola 4.3.

### 4.1.2 Výstupní soubory

V těchto souborech jsou uváděny výsledky optimalizačního výpočtu. Hlavním souborem je `res.dat`. Jde o soubor, do kterého optimalizační program uloží vypočítané rozložení modulů v kontejneru. Soubor poté využívá zobrazovací část aplikace. Struktura tohoto souboru je popsána v tabulce A.6.

Mimo tento soubor se při výpočtu uložení materiálu vytvoří ještě další tři soubory, do kterých je rozdělen materiál ze vstupního souboru `EXP_H_TETRIS.txt`. Jedná se o soubory `used.dat`, `unused.dat` a `error.dat`. Jak název napovídá, soubor `used.dat` slouží k uložení informací o materiálu, který byl použit při výpočtu optimalizace, a je již uložen v nějakém kontejneru. Soubor slouží především pro kontrolu. S tímto materiálem se již nesmí v dalším cyklu optimalizace počítat.

Druhý soubor `unused.dat` obsahuje naopak materiál, který nebyl vhodným způsobem naložen do kontejnerů, a musí tedy být zahrnut do výpočtu v dalším cyklu. Data se tam prakticky ukládájí průběžně a na konci výpočtu aktuálního cyklu optimalizace jsou veškeré materiály z tohoto souboru nakopírovány zpět do vstupního souboru `EXP_H_TETRIS.txt`.

Poslední soubor z této skupiny `error.dat` obsahuje materiál, pro který nebyly při načítání dohledány všechny potřebné informace, a nešlo ho tedy zařadit do výpočtu. Většinou se jedná o doplňkový materiál. Tento problém si budou pracovníci řešit sami. Do dalšího cyklu výpočtu se tak tento materiál již nezařazuje.

### 4.1.3 Konfigurační soubory

Aplikace obsahuje dva konfigurační soubory. První, `config.xml`, slouží přímo optimalizačnímu programu. Obsahuje pouze pořadové číslo kontejneru, pro který proběhne další cyklus výpočtu. Po domluvě bylo zvoleno 7-mi místné číslo, tedy začátek na 1 000 000.

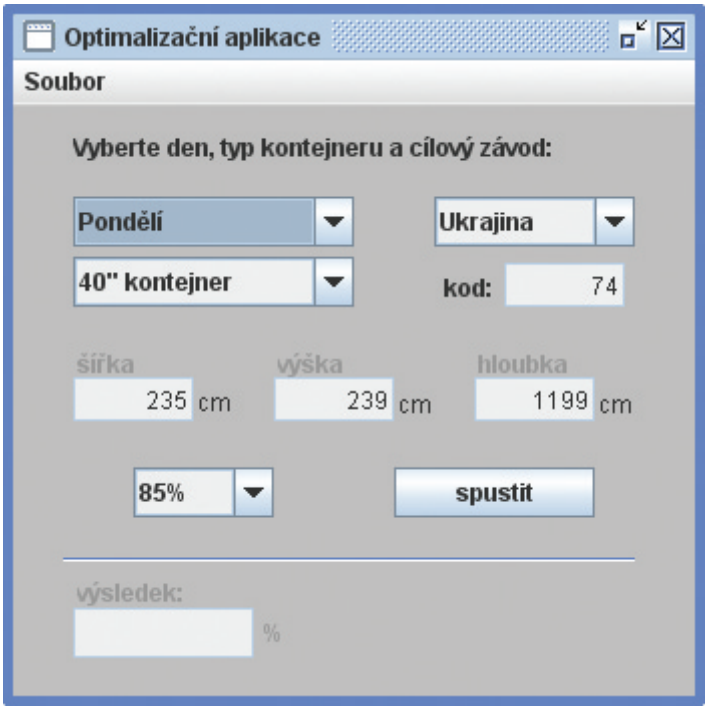
Druhý konfigurační soubor `bconfig.dat` slouží k uložení všech dosud nevyexpedovaných kontejnerů. Struktura tohoto souboru je shodná se souborem `res.dat`. Rozdělení do dvou souborů je především z důvodu bezpečnosti. Zatímco `res.dat` používají dva programy, kde jeden do něho zapisuje a druhý z něho čte, `bconfig.dat` slouží pouze zobrazovací části aplikace.

## 4.2 Grafické uživatelské rozhraní pro získání požadavků pro optimalizaci

Tento program slouží ke snadnému získání kritérií pro výpočet optimalizace. Hlavní jeho funkcí je spustit se správnými parametry probram `b2c.exe`, který přímo provádí výpočet. Ukázka rozhraní je uvedena na obrázku 4.2. a popis jednotlivých částí je v následující sekci 4.2.1 Po spuštění, které lze jednoduše provést přes zástupce `b2c.lnk`, je otevřeno okno o velikosti 350 × 350 pixelů s názvem *Optimalizační aplikace*.

### 4.2.1 Přehled jednotlivých částí

V horní části okna je umístěno menu, které obsahuje ovšem pouze tlačítko pro uzavření. Jiné možnosti nejsou potřebné. Menu jsem se rozhodl ponechat kvůli uživatelům, kteří jsou



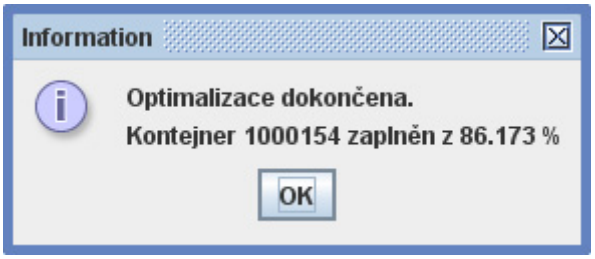

Obrázek 4.2: Screenshot rozhraní pro získání požadavků pro optimalizaci

na tuto možnost ukončení programů zvyklí. Dále samotné okno obsahuje tři rolovací lišty. První skrývá všechny dny v týdnu (Pondělí až Neděle). Zvolená kolonka označuje den, pro který se má optimalizace vypočítat. Lze vypočítat optimalizaci dopředu, maximálně však o týden, jelikož program nerozlišuje datum, ale pouze den. Rozlišení data je v tomto případě zbytečné, protože balící plán je vytvářen maximálně několik dní dopředu, tudíž více než na týden dopředu by stejně nebyl znám potřebný materiál.

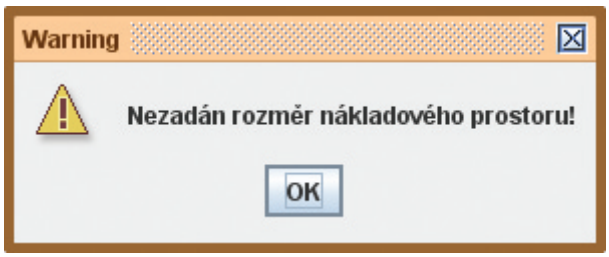
Druhý rolovací seznam obsahuje všechny (dosud známé) montážní závody, do kterých se může daný kontejner posílat. Pokud je vybrán závod z nabídky, v políčku pod listboxem se zobrazí kód závodu (např. “74” pro Ukrajinu). Pokud je třeba zadat jiný závod, lze v seznamu zvolit možnost “... jiný závod”. Již zmiňované políčko pod rolovací lištou se zaktivní a lze do něho napsat kód závodu ručně.

Poslední seznam obsahuje několik běžných typů kontejnerů a vagónů, které se používají pro přepravu materiálu. Tato rolovací lišta je propojena se třemi políčky pod ní, takže po zvolení nějaké možnosti se zobrazí i velikost daného nákladového prostoru. Pokud v seznamu není potřebný kontejner, lze zadat opět možnost “... jiný rozměr” a do políček pod seznamem vyplnit požadovanou velikost ručně. Velikost nákladového prostoru je v centimetrech, to platí i pro zadávání velikosti ručně! Byla také stanovena horní hranice velikosti nákladového prostoru a to na 100 m ve všech třech rozměrech, což je více než postačující. Pokud není zadaná hodnota jakéhokoliv rozměru v intervalu $0-10\,000$ cm nebo číslo obsahuje chybný znak, například písmeno, program upozorní uživatele varovnou hláškou 4.4. V takovém případě nelze pokračovat ve výpočtu, dokud není chyba odstraněna.

Po vyplnění všech políček lze stisknout tlačítko “Spustit”, které spustí samotný optimalizační program se zadanými parametry 4.3.1. Během výpočtu nelze jakkoliv program ovládat. Na úspěšné ukončení výpočtu upozorní hláška 4.3, která obsahuje číslo kontejneru




Obrázek 4.3: Hláška při ukončení výpočtu



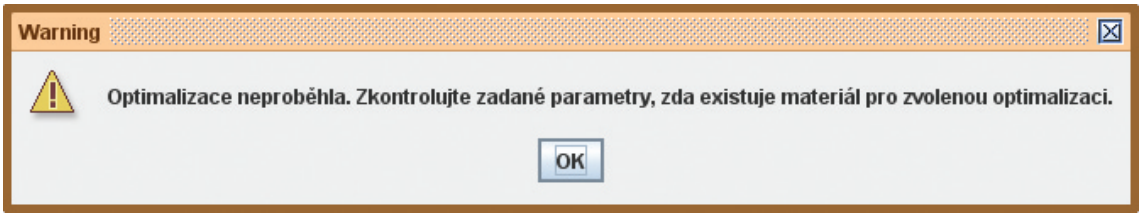

Obrázek 4.4: Varovná hláška

a jeho procentuální zaplnění. Daná optimalizace je již uložena v souboru `res.dat` a lze ji zobrazit.

Pokud všechny zadané parametry jsou syntakticky správně, avšak z nějakého důvodu nelze rozložení vypočítat, zobrazí se informační hláška 4.5. K tomuto může dojít například, pokud není dostatečné množství materiálu, či pokud nelze materiál vhodným způsobem uspořádat.

## 4.3 Optimalizační program

Tato podkapitola se zabývá jádrem celého projektu, a to programem, který vypočítává rozložení modulů ve zvoleném nákladovém prostoru. Mimo způsobu, jakým má být program spuštěn, zde naleznete základní principy, jak program pracuje.




Obrázek 4.5: Informační hláška při neprovedení optimalizace

### 4.3.1 Vstupní parametry

Optimalizační program `b2c.exe` je spouštěn s mnoha parametry. Většina z nich jsou požadavky uživatelů na provedení výpočtu. Jiné jsou nastaveny grafickým rozhraním na stále stejnou hodnotu. Takové parametry na první pohled nejsou vůbec potřebné, dodány tam však byly z důvodu možnosti dalšího využití a rozvoje samotného optimalizačního programu (viz. kapitola 6.2).

Po spuštění jsou všechny parametry přečteny funkcí `getParams()` a jsou uloženy do struktury `params` tak, aby k nim byl během celého procesu výpočtu možný přístup. Pokud pro spuštění nepoužijeme grafické uživatelské rozhraní, lze program spustit také s parametrem `-h`, tzn. `./b2c.exe -h`. Tím se vypíše obrazovka s nápovědou, jak lze jednotlivé parametry zadat a co znamenají (viz. tabulka 4.1).

Příklad použití: `b2c.exe -pocet_dnu (PO-NE) mira_uspesnosti -kontrola_vysledku cislo_zavodu cislo_kontejneru velikost_x velikost_y velikost_z -pocet_kontejneru`.

| Parametr | Popis |
|---|---|
| –pocet_dnu | počet dnů, které se mají vypočítat |
| (PO-NE) | seznam požadovaných dnů |
| mira_uspesnosti | určuje, kdy lze ukončit výpočet |
| –kontrola_vysledku | požadavek, zda se má před ukončením odsouhlasit výsledek zaplnění |
| cislo_zavodu | číslo závodu, kam se materiál posílá |
| cislo_kontejneru | pořadové číslo kontejneru |
| velikost_x | šířka nákladového prostoru v cm |
| velikost_y | výška nákladového prostoru v cm |
| velikost_z | hloubka nákladového prostoru v cm |
| –pocet_kontejneru | počet kontejnerů, které se mají vypočítat |

Tabulka 4.1: Popis vstupních parametrů

Jak je patrné z požadavků Škody, vypočítává se v danou chvíli pouze jeden kontejner a výsledek se nekontroluje. Právě v těchto oblastech je předpoklad dalšího vývoje aplikace. Program s těmito možnostmi však již pracuje a je možné jej plně využívat např. při testování či výzkumu.

### 4.3.2 Načtení a uchovávání dat během výpočtu

Po zpracování parametrů dojde k načtení informací o materiálu funkcí `readData()`. Během výpočtu musí být všechny informace neustále k dispozici. Byly proto vytvořeny dva seznamy, jejichž struktury včetně jednotlivých složek jsou uloženy v hlavičkovém souboru `b2c.h`.

První seznam obsahuje všechny moduly, se kterými lze při výpočtu počítat. Každý modul je zde reprezentován strukturou, která je znázorněna v tabulce 4.2 včetně popisu jednotlivých položek. Uchovávány tak jsou veškeré potřebné informace o materiálu. Pro zrychlení výpočtu jsou položky v seznamu řazeny podle velikosti od největších po nejmenší. Bylo také nutné rozhodnout, jaké datové typy budou postačující pro dané položky. U většiny číselných hodnot byl použit `unsigned int`, který svým rozsahem je více než dostačující, např. pro závěsku či váhu modulu.

| Položka struktury | Popis |
|---|---|
| unsigned int zaveska | závěska slouží k identifikaci modulu |
| char matnb[19] | kód materiálu v modulu |
| tsize size | velikost modulu, vnořená struktura obsahuje tři rozměry |
| unsigned int weight | váha modulu včetně materiálu |
| unsigned int max_load | nosnost modulu |
| int rotation | příznak otočení modulu |
| tsize point | umístění v kontejneru |
| unsigned int capacity | objem modulu |
| int used | příznak použití modulu při výpočtu |
| char modul[10] | označení modulu |
| int counting | příznak započítání do výsledku |
| char fullstring[64] | celý řetězec s parametry pro snadné vypisování |
| int pocetbal | počet balení materiálu |
| twrap wrap | obsahuje počet a název vložených modulů |

Tabulka 4.2: Struktura box

Druhý seznam obsahuje ukazatele na položky prvního seznamu, které jsou v danou chvíli použity v kontejneru (viz. sekce 4.3.3). Celý výpočet je prováděn rekurzí, tudíž je nutné znát pořadí, v jakém se moduly vkládají do kontejneru, aby v případě neúspěšného výpočtu mohly být zase jednotlivě odebírány.

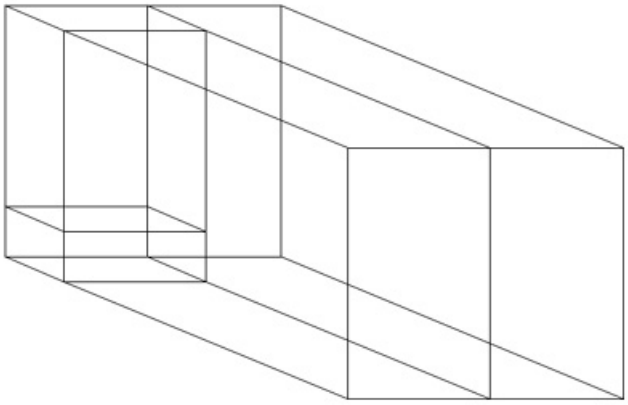
### 4.3.3 Princip výpočtu uložení materiálu

V této sekci bude popsán základní princip výpočtu optimalizace. Po zpracování parametrů a vytvoření seznamu s informacemi o jednotlivých modulech, zavolá hlavní funkce `main()` funkci `optimalization()`. Této funkci je jako parametr předán mimojiné ukazatel na datový typ `tspace`, ve kterém jsou uloženy informace o nákladovém prostoru, do kterého se optimalizace vypočítává.

Funkce v hlavním cyklu "bere" jednotlivé položky ze seznamu (4.2) a "vkládá" je do daného prostoru (do levého spodního rohu). Zda lze opravdu modul do prostoru vložit, kontroluje funkce `checkbox()`. Pokud může dojít k vložení modulu do prostoru, vytvoří se další tři imaginární nákladové prostory, alokují se tři datové typy `tspace`, a vyplní se hodnotami. Jeden nad daným modulem, druhý vedle a třetí před ním (obrázek 4.6).Jak již bylo řečeno, algoritmus funguje na principu rekurze, tudíž v tomto místě funkce `optimalization()` volá sama sebe. V podstatě se volá právě třikrát, jako parametr ji je totiž předán pokaždé jeden z nově vytvořených prostorů. Takto dojde k rozdělení kontejneru na malé části, do kterých jsou vkládány jednotlivé moduly.

Pokud ve vrchních patrech takto vytvořené pyramidy dojde k dostatečnému zaplnění kontejneru, vrací funkce do vrstvy pod sebou objem zaplněného prostoru. Pokud k tomuto nedojde, postupně se odebírají z prostoru moduly, a funkce zkouší do daného prostoru vložit jiný modul a výpočet opakovat. Takto algoritmus probíhá ve smyčce dokud se nedosáhne v celém kontejneru dostatečného zaplnění (zadaného uživatelem).

Již zmiňovaná funkce `checkbox()` navíc během testování, zda je modul možné vložit do daného prostoru, také zkoumá, zda se nejedná o GLT modul. Tedy o modul, který lze stohovat podle specifických pravidel (viz. tabulka A.1). Pokud ano, vyzkouší možnosti



Obrázek 4.6: Rozložení prostorů v kontejneru

“podložení” modulu.

Jak jsou jednotlivé moduly ukládány do kontejneru, vytváří se na ně ukazatele a vkládají se do druhého seznamu. Takto lze poté jednoduše moduly zase odebírat a máme neustále přehled, které moduly jsou již do optimalizace zahrnuty a které ne.

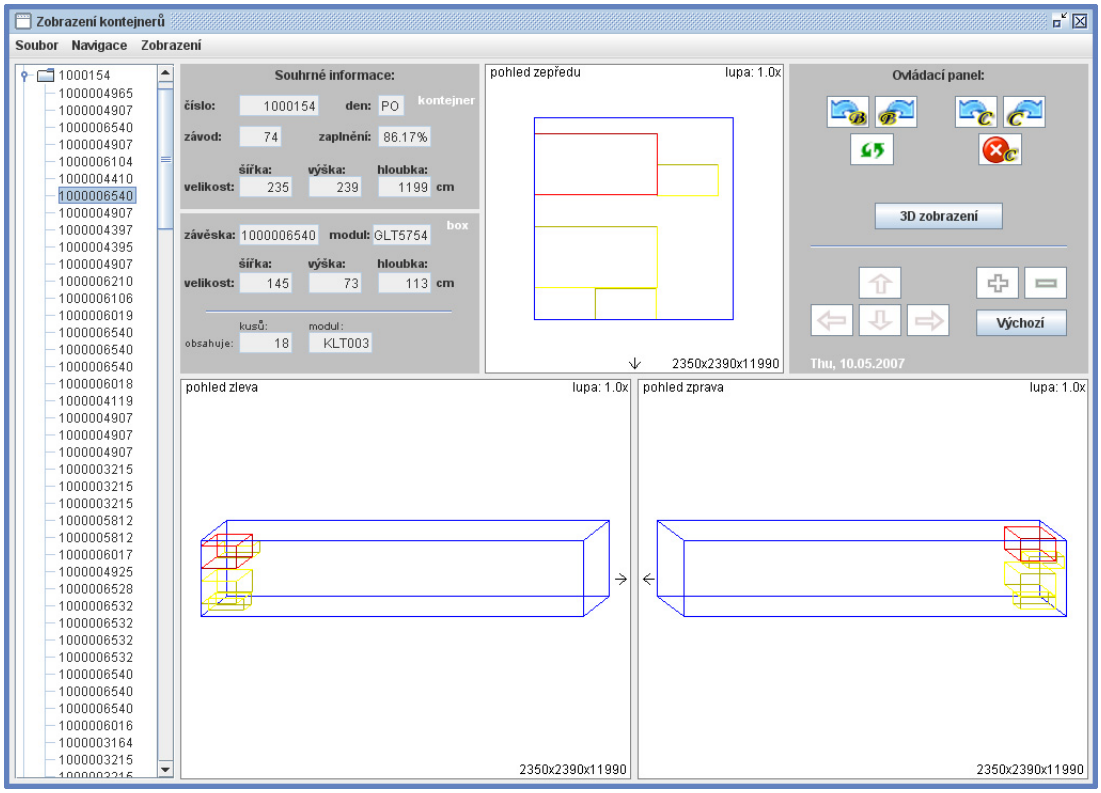
### 4.3.4 Uložení výsledku a ukončení programu

Pokud výpočet proběhne úspěšně, tzn. míra zaplnění přesáhla požadovanou hranici, je nutné všechna data vhodně uložit. To má na starosti funkce `saveResult()`. Nejdříve je nutné však přepočítat použitý materiál. Prochází se oba seznamy a do souborů `used.dat` a `unused.dat` jsou rozdělovány jednotlivé moduly s materiálem. K tomu slouží funkce `countingBoxes()`.

Dříve popsaný způsob ukládání položek do druhého seznamu s ukazateli naznačil, že takovéto pořadí modulů je vhodné pro výpočet. Ovšem při zobrazení by vznikl značný chaos. Je proto nutné moduly přeskládat.Následovat po sobě musí tak, jak by se optimálně skládali do kontejneru, tedy odzadu dopředu a odzdola nahoru. To zařizuje funkce `rangeResults()`, která vytvoří pomocný seznam a moduly přerovná. Nyní již můžeme vypočítané rozmístění modulů v kontejneru uložit do souboru `res.dat`. Nejdříve jsou uloženy všechny informace o nákladovém prostoru, poté seznam všech modulů a jejich umístění. Průběžně funkce uvolňuje paměť alokovonanou složkami seznamu. Na konci se uvolní veškeré alokované seznamy. Program ještě na standartní výstup vytiskne míru zaplnění, kterou zobrazí grafické uživatelské rozhraní, a bezpečně se ukončí.

## 4.4 Program pro zobrazení kontejnerů a jejich naložení

Tato podkapitola se zabývá zobrazením vypočítaného rozložení modulů v kontejneru. Důraz byl kladen především na přehlednost, aby pomocí programu bylo možné přímo provádět nakládku kontejneru. Pro jednoduché spuštění byl vytvořen odkaz `b2cII.lnk` v hlavním adresáři. Po spuštění programu se otevře hlavní okno, které je znázorněno na obrázku 4.7. Velikost okna je nastavena na $1024 \times 735$ pixelů, tedy na takovou velikost, aby se na monitorech s nejpoužívanějším rozlišením $1024 \times 768\,px$ zobrazilo přes celou obrazovku. Z důvodů velikosti jednotlivých vnořených částí, které jsou popsány v části 4.4.1, nelze velikost hlavního okna upravovat.




Obrázek 4.7: Screenshot programu pro zobrazování naložení kontejnerů

### 4.4.1 Přehled jednotlivých částí

Okno s názvem "Zobrazení kontejnerů" je rozděleno do čtyř hlavních částí. Vlevo je umístěn stromový seznam s aktuálními kontejnery, které ještě nebyly vyexpedovány. Všechny jsou označené svými identifikačními čísly. Po kliknutí na číslo kontejneru se položka rozvine a objeví se seznam všech modulů, které po výpočtu optimalizace kontejner obsahuje. Toto je nejjednodušší způsob ovládání. Aktivní prvek, kontejner nebo modul, jsou označeny modrým podkladem.

Další část tvoří panel s nadpisem "Souhrné informace". V něm se zobrazují všechny důležité informace o právě aktivním prvku v již zmiňovaném seznamu. Rozdělen je do dvou částí, v horní se zobrazují informace o kontejneru a ve spodní části o aktuálním modulu. Pokud je aktivní prvek v seznamu přímo kontejner, ve spodní části se samozřejmě nic nezobrazuje. U kontejnerů jsou zde informace o ID čísle, dnu, kdy má být odeslán, kódu závodu, kam je posílán, a jeho velikosti v centimetrech. Spodní část, zobrazující informace o aktivním modulu, naopak zobrazuje závěsku, označení modulu, velikost modulu, počet a druh vložených modulů, pokud nějaké obsahuje.

Jednou z nejdůležitějších částí celého programu jsou okna, kde se naložení kontejneru přímo zobrazuje. Jsou tři a jsou na sobě plně závislá. Každé však zobrazuje kontejner z jiného pohledu. První menší okno je umístěno uprostřed v horní části a zobrazuje kontejner zepředu, tzn. ze strany, kde jsou dveře. Další dvě větší okna jsou rozmístěna v dolní části hlavního okna a ukazují kontejner zprava, resp. zleva. Kde jsou dveře kontejneru znázorňuje šipka na straně. Všechna okna navíc ve svých okrajových částech obsahují informace o směru pohledu, zvětšení a velikosti kontejneru samotného.

Kontejnery jsou zde znázorněny modře, naložené moduly v nich žlutě a poslední (aktivní) modul červeně. Pokud bychom najednou zobrazili všechny moduly v kontejneru, které jsou již naložené, nebylo by z obrázku možné téměř cokoliv vyčíst. Zobrazují se tudíž pouze moduly, které jsou bezprostředně nejblíže aktivnímu modulu. Pokud je modul například ve vyšší vrstvě, zobrazí se s ním pouze moduly pod ním.

Pro ještě lepší znázornění přesného umístění modulu v kontejneru, obsahuje program další dvě funkce. Jedna z nich je možnost zvětšení pohledu. Velikost prvků v oknech se poté zvětší, tudíž zobrazí větší detail. Implicitně je pohled po zvětšení zarovnán na aktivní modul, tudíž druhou důležitou funkcí je možnost "pohybu" v zobrazovacích oknech. Lze se pohybovat jednak po ose $x$, tzn. doprava a doleva, a také ve směru osy $y$, tudíž nahoru a dolů. K použití těchto funkcí slouží buď tlačítka v ovládacím panelu (viz. dále) nebo nabídka menu v záložce `Zobrazení`. Jediné okno nepodléhající těmto úpravám je horní, které zobrazuje čelní pohled.

Poslední částí je již zmiňovaný ovládací panel. Obsahuje tlačítka pro posun v seznamu s kontejnery (a to jak na další modul, tak na další kontejner a zpět). Zda se jedná o možnost týkající se kontejnerů nebo modulů, je zřejmé z malého $c$ (z anglického *container*), resp. $b$ (z anglického *box*). Dále je zde tlačítko pro aktualizaci seznamu, tzv. "Refresh". Po stisknutí program zkontroluje soubor `res.dat`, zda od doby spuštění nedošlo k výpočtu dalšího kontejneru. Pokud ano, zařadí nový kontejner na konec aktuálního seznamu kontejnerů a lze ho zobrazit. Tlačítko s označením "Uzavření (vymazání) kontejneru" slouží k ukončení balícího cyklu daného kontejneru. Program kontejner vyřadí ze seznamu a již ho dále neukládá. Tudíž daný kontejner již nelze zobrazit ani se k němu nijak vracet. Ve spodní části panelu jsou tlačítka umožňující již zmiňovaný pohyb a zvětšování. K jednoduchému návratu slouží tlačítko "Výchozí", po jehož stisknutí jsou kontejnery opět vykresleny v původní velikosti a pozici.

Všechny tyto možnosti, které obsahuje ovládací panel, lze rovněž nalézt v menu. To je tradičně umístěno v horní části okna a je rozděleno do tří základních položek: `Soubor`, `Navigace` a `Zobrazení`.

### 4.4.2 3D zobrazení

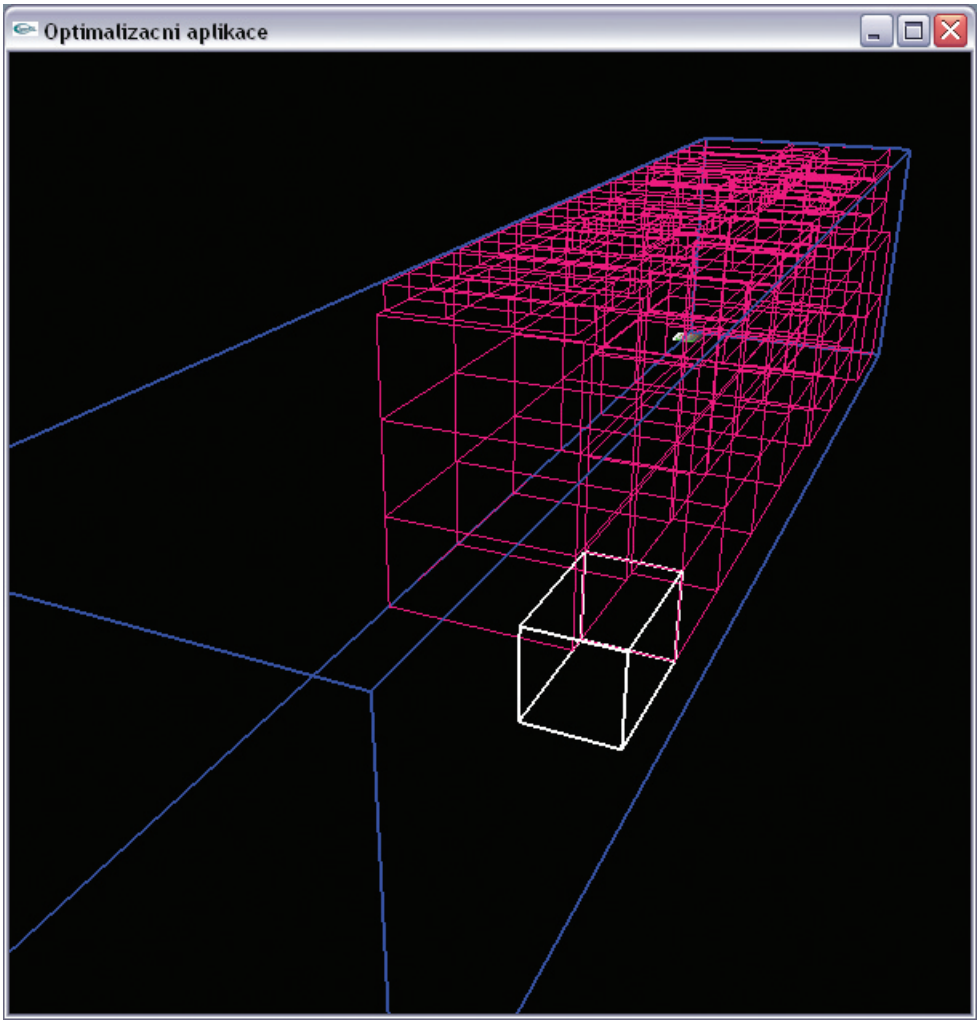
Velkým specifikem celého programu je možnost zobrazení ve 3D síťového modelu kontejneru. Jde o samostatný program spustitelný pomocí tlačítka v ovládacím panelu "3D zobrazení". Škoda původně plánovala zařadit tento program jako součást svých aplikací a vlastní zobrazovací program, jak je popsán výše, vůbec nevytvářet. Změna je ale život, návrh pozdějšího zobrazovacího programu již s možností 3D zobrazení nepočítal, především pro její obtížnost orientace při běžném používání. Avšak zřejmě bych aplikaci ochudil, kdybych možnost 3D zobrazení nezahrnul vůbec.

Program je napsán v OpenGL. Při zobrazení využívá soubor `bconfig.dat`, do kterého zobrazovací program ukládá seznam kontejnerů. Při spuštění zobrazí pouze jedno okno s daným kontejnerem. Ovládání se provádí pomocí klávesnice, jak je popsáno v tabulce 4.3. Pouze otáčení kolem zadního dolního bodu kontejneru se provádí pomocí stisknutí a pohybu myši.

Jako parametr při spuštění program očekává číslo kontejneru, které se má načíst a index aktivního modulu. Zobrazí tak kontejner shodně, jako všechna tři okna zobrazovacího programu. Ukázka programu je znázorněna na obrázku 4.8. Při jakékoliv činnosti v hlavním okně zobrazovacího programu, je okno 3D zobrazení uzavřeno.

| Klávesa | Popis |
|---|---|
| f | fullscreen |
| g | velikost okna $500 \times 500\,px$ |
| u | vložení dalšího modulu |
| i | odebrání posledního modulu |
| p | vložení všech modulů |
| o | vyprázdnění kontejneru |
| d | pohyb doprava |
| a | pohyb doleva |
| w | pohyb nahoru |
| s | pohyb dolů |
| q | uzavření okna |

Tabulka 4.3: Ovládání programu pro 3D zobrazení




Obrázek 4.8: Zobrazování pomocí 3D síťového modelu

# Kapitola 5

# Implementace a testování

Tato kapitola se zabývá popisem implementace celé aplikace, použitých nástrojů a problémů návrhu, které bylo nutné vyřešit či které na vyřešení ještě čekají. Průběžně bude zmíněn postup při testování jednotlivých částí, který bude na závěr shrnut.

Grafické rozhraní pro získání požadavků i zobrazovací program jsou napsány v jazyce Java. To přináší i určitá opatření, která jsou potřebná pro jejich bezproblémové spuštění, jako je např. nutnost instalace Java Virtual Machine. To je v podstatě software, který vystupuje jako hardwarové zařízení a které převádí bytový kód (který vytváří překladač jazyku Java) do nativních instrukcí pro daný hostitelský počítač. Zmíněný bytový kód (nebo také pseudokód) je zjednodušeně optimalizovaná sada instrukcí. Velkou výhodou je přenositelnost těchto bytových kódů [3]. Oba programy psané v Javě byly testovány v prostředí Java 2, Standart Edition SKD (JDK 1.4).

Optimalizační program je poté napsán pomocí standartních knihoven v jazyku C a program umožňující 3D zobrazení naložení kontejnerů poté ještě využívá knihovny OpenGL. Oba tedy využívají překladač, který je závislý na typu procesoru a nezřídka i na operačním systému. Překladač ale vytvoří spustitelný kód, který je soběstačný. Nevýhodou je, že takto vytvořené programy nelze přenášet do jiných prostředí. Na snadě je tedy otázka, proč nejsou všechny části aplikace psané stejným programovacím jazykem. Důvod je prostý, původně neměl mít optimalizační program žádné grafické rozhraní, přes které by se zadávaly parametry. Tedy bylo jedno, v jakém jazyce bude optimalizační program napsán a jazyk C se zdál postačující.

## 5.1 Adresářová struktura

Celá aplikace je vložena do jediného adresáře. Ve Škodě byl pro aplikaci určen prostor v adresáři `\\skdambveot\eota$\eot6$\hala_U33`. Přímo zde jsou dva zástupci `b2c.lnk` a `b2cII.lnk`. První spustí grafické rozhraní pro zadávání požadavků, jak je popsáno v podkapitole 4.2. Druhé pak zobrazovací program popsaný v podkapitole 4.4.

Ve složce `data` jsou poté umístěny všechny programy včetně souborů potřebných k jejich spuštění. Zatímco ve druhé složce `sources` se nacházejí zdrojové kódy.

## 5.2 Grafické uživatelské rozhraní pro získání požadavků pro optimalizaci

Program byl implementován podle návrhu popsaném v podkapitole 4.2. V této podkapitole se detailněji podíváme na způsob, jakým byl program vytvořen především z pohledu programátora. Jak již bylo řečeno, je celý napsán v Javě. Grafické komponenty byly vytvořeny především pomocí knihovny Swing definované v balíčku `javax.swing`. Program tvoří tři třídy. Třída `Main` obsahuje pouze funkci `main()` a slouží k zobrazení hlavního okna pomocí funkce `run()`. Hlavní třídou je tak `BFrame`, která se pak stará o vytvoření a běh tohoto okna.

Na začátku vytvoří hlavní rámec (frame), do kterého umístí všechny komponenty jak jsou popsané v sekci 4.2.1. Rolovací seznamy jsou vytvářeny komponentou `JComboBox`, zatímco políčka pro zadávání například cílového montážního závodu jsou vytvářeny komponentou `JTextField`. Po vytvoření tototo okna je nejdůležitější správně zachytávat vzniklé události. K tomu slouží metoda `actionPerformed(ActionEvent e)`, která například při stisknutí tlačítka "Spustit" volá funkce, které vedou ke spuštění optimalizační aplikace. Jde především o funkci `checkExe()`, která nejprve kontroluje zadané parametry. Pokud nastane nějaká chyba, vypíše chybovou hlášku. Poté spustí optimalizační program a čeká na odpověď.

Poslední třídou je `BConfig`. Konstruktor třídy na začátku načte konfiguraci, tedy především číslo kontejneru, jehož naložení bude optimalizováno. Dále se také stará o uložení aktuální konfigurace například při ukončení programu.

## 5.3 Optimalizační program

Optimalizační program byl implementován podle návrhnu popsaném v podkapitole 4.3 včetně všech základních principů. Jsou zde nastíněny i funkce a ovládání programu. V této podkapitole bych se proto více věnoval "bolavým místům", které při implementaci vznikly.

Největší z nich zřejmě je, že ačkoliv kontrola přetížení jednotlivých modulů v algoritmu je implementována, lidé ze Škody ještě nezjistili váhu všech materiálů. Nemohou je tedy uvádět u všech položek v souboru `EXP_H_TETRIS.txt`, a tudíž je tato funkce z programu zatím vyčleněna. Program tak vypočítává rozložení pouze s ohledem na velikost modulů a možnost jejich stohování. Tato skutečnost se samozřejmě odrazila především na době výpočtu. Lze předpokládat, že po začlenění této funkce zpět vzroste čas, který je k výpočtu potřeba. Nejde ani tak o samotnou funkci, která přetížení kontroluje, ale především o to, že bude vyhovovat pro dané umístění menší procento modulů, čili se vhodný modul bude hledat déle.

Tím narážíme na další problém. Jak stanovit dobu, po kterou je možné na výsledek ještě čekat? Omezit program časově by bylo zřejmě krátkozraké, jelikož by poté na každém typu procesoru provedl jiný počet rekurzí. Omezen je tedy počet rekurzí, který algoritmus provede. V globální proměnné `timer` je inkrementována hodnota pokaždém volání funkce `optimalization()`, zároveň je v tuto chvíli i daná proměnná porovnávána s proměnnou `TIMER`, ve které je uložena hodnota udávající maximální počet rekurzí. V našem případě stanovena na 20 000 000 a v případě překročení této hodnoty provede program nezbytné kroky a ukončí se. Jak je vidět při testování z tabulky A.8, kde je přehled výsledků testování, je to více než postačující hodnota. Po začlenění kontroly přetížení modulů bude nutné hodnotu opět uvážit, ale myslím si, že i poté bude 20 mil. postačovat.

## 5.4 Program pro zobrazení kontejnerů a jejich naložení

Tato část aplikace je zřejmě největší, co se týče řádků kódu. Napsaná je v jazyce Java. Obsahuje celkem 16 tříd, které se starají o bezproblémový běh programu, a jsou všechny umístěny v balíku `b2cii`, což odpovídá i organizaci adresáře se zdrojovými kódy. Vstupní bod do programu je třída `Main`, která obsahuje metodu `main()`. Jak bylo popsáno v podkapitole 4.4, hlavní okno aplikace obsahuje několik prvků. Každá komponenta je poté ovládána samostatnou třídou, která obsahuje všechny potřebné metody.

Než si však ukážeme třídy, které jsou v programu "vidět", je důležité popsat dvě třídy, se kterými ostatní neustále pracují. Jedná se o třídy `BContainer` a `BBox`. Instance těchto tříd tvoří uzly stromového seznamu s kontejnery a obsahují všechny potřebné informace o kontejneru či modulu.

Hlavní třídou programu pak je `BMainWindow`. Její konstruktor vytvoří nejprve hlavní okno a upraví jeho nastavení. Deklaruje proměnné objektových typů, pomocí operátoru `new` vytvoří a přidá postupně všechny objekty, které si dále jednotlivě rozebereme. Jednou z nejdůležitějších funkcí této třídy je právě propojení oněch objektů. Pokud v seznamu kontejnerů dojde k zachycení události, například posunu aktivního prvku, tato třída zajistí, aby se daná změna projevila i v zobrazovacích oknech. K tomu obsahuje třída metody `setCont()` na nastavení aktivního kontejneru a `setBox()` na vykreslení modulů. Třída se dále také metodou `saveConfig()` stará o ukládání aktuálního stavu seznamu kontejnerů či metodou `Refresh()` o zjištění nově vypočítaných kontejnerů.

A nyní si projdeme jednotlivé objekty. Třída `BMenuBar` vytváří lištu menu. Z deklarace třídy je patrné, že "rozšiřuje" třídu `JPanel` z knihovny Swing. Konstruktor vytvoří novou instanci třídy JMenuBar a postupně do ní vloží všechny záložky. Pomocí rozhraní `ActionListener` jsou zde zachytávány události, tedy v našem případě kliknutí tlačítka myši na nějakou nabídku v menu. O "obsloužení" takovéto události se stará metoda `actionPerformed()`.

Další komponentou v programu je stromový seznam kontejnerů obsahující i dané moduly. O vytvoření tohoto objektu se stará konstruktor třídy `BTreeStruct`. Na začátku dojde k načtení dat ze souborů `bconfig.dat` a `res.dat` pomocí stejné metody `readData()`, protože oba soubory mají shodnou strukturu. Jednotlivé řádky souborů se musí rozdělit na jednotlivá slova reprezentující různé informace. K tomu je využívána instance třídy `StringTokenizer` a její metoda `hasMoreTokens()`. Postupně jsou data ukládána do datových typů `BContainer` či `BBox` a ty jsou použity k vytvoření uzlů stromu (datový typ DefaultMutableTreeNode), které jsou nakonci ukládány do instance třídy JTree. Uzly reprezentující kontejnery obsahují tak poté ještě tzv. "listy", které reprezentují jednotlivé moduly v kontejneru. Tento "strom" je poté vložen přímo do objektu BTreeStruct, čímž se zobrazí daný stromový seznam. Třída `BTreeStruct` dále ještě obsahuje metodu `changeValue()`, která obsluhuje vzniklé události, a také řadu metod, které se starají o posouvání aktivního prvku či smazání uzlů ve stromové struktuře.

Daný aktivní prvek je poté důležitý pro celý program, jelikož se jedná o aktuální modul, který se má naložit do kontejneru (nebo přímo o samotný kontejner). Informace o takovémto prvku jsou zobrazovány v informačním panelu, který je tvořen dvěma částmi. První, která zobrazuje informace o kontejneru, je objekt třídy `BContPanel`. Druhá, zobrazující modul, je poté instance třídy `BBoxPanel`. V podstatě mají stejnou úlohu. V konstruktorech jsou do instance třídy JPanel, která je umístěna v hlavním rámci, vloženy všechny komponenty a nastaveny vlastnosti zobrazení, jak je uvedeno v sekci 4.4.1. Důležitou roli zde ale hrají metody `setCont`, resp. `setBox`, které do políček v panelu vkládají příslušný text, který se

týká aktuálního prvku (uzlu) v seznamu kontejnerů. Volány jsou z hlavní třídy programu `BMainWindow` vždy, když dojde k nějaké změně.

Nejdůležitější část programu jsou okna, ve kterých se znázorňuje naložení kontejnerů. Každé okno je popsáno v jedné třídě. Horní, které znázorňuje naložení kontejneru z čelního pohledu, je instancí třídy `BCenterView`. Spodní okna s bočními pohledy pak třídy `BRightView`, resp. `BLeftView`. Nejdříve si popíšeme horní okno, které je z nich nejjednodušší, zobrazuje totiž naložení pouze ve 2D prostoru. Konstruktor třídy pouze nastaví pozadí okna na bílou barvu. Vykreslovací metody jsou metody třídy Canvas, kterou třída `BCenterView` rozšiřuje. Jedná se především o metodu `paint(Graphics g)`, která nejprve zjistí, jaký kontejner se má vykreslit. Do proměnné `pomer` uloží poměr šířky kontejneru k šířce okna (nebo výšky kontejneru k výšce okna v závislosti na tvaru kontejneru), toto důležité číslo se poté používá k určení velikosti všech modulů uvnitř. Dále prochází pozpátku stromový seznam kontejnerů a od aktivního modulu vykresluje všechny moduly naložené předním. Tím docílíme lepší názornosti, kam má daný modul přijít. Moduly se vykreslují pomocí metody `draw2DBox`. Metoda `draw2DCont` poté slouží k vykreslení kontejneru, metody `vykreslitUdaje()` a `vykreslitVelikost()` k zobrazení údajů o vlastnostech do okrajů okna.

Další dvě již zmiňované třídy `BLeftView` a `BRightView` jsou v mnohém hodně podobné třídě `BCenterView`. Princip zobrazování je shodný, pouze pomocí 2D primitiv (většinou jednoduchých čar) je zobrazována 3D perspektiva. Kontejner je navíc vykreslován nadvakrát, zadní strana se zobrazuje před vykreslováním modulů, přední až po něm. Aby nedošlo k nežádoucím překryvům. Třídy navíc obsahují metody pro posun zobrazovaných objektů všemi směry a metody `zmensit()` a `zvetsit()` pro úpravu velikostí. Oproti normálnímu stavu lze zmenšit objekt na 50 %. Větší zmenšení nemá z pohledu přehlednosti smysl. Míra zvětšení omezena není, přesto nějaké velké zvětšení rovněž nemá význam. Po každé úpravě a přepočítání rozměrů se znovuvykreslení provádí metodou `repaint()`.

Poslední viditelnou částí programu je ovládací panel v pravé horní části hlavního okna. Jde o instanci třídy `BControlPanel`. Konstruktor této třídy se stará o vložení všech komponent, viz. sekce 4.4.1. Důležitou roli zde hraje metoda `actionPerformed()`, která rozlišuje jednotlivé události a volá příslušné metody jiných tříd. Tím je zajištěno, že pomocí těchto tlačítek lze ovládat celou aplikaci. Třída mimojiné také zajišťuje spouštění již zmiňovaného programu pro zobrazení síťového 3D modelu naložení kontejneru s možností pohybu v tomto prostoru, viz. sekce 4.4.2. Vedlejší možností je také zobrazení dne a datumu, které je v pravém dolním rohu panelu.

### 5.4.1 Program pro 3D zobrazení

Jak již bylo řečeno, program je psán v jazyce C s využitím knihovny OpenGL. OpenGL (Open Graphics Library) je nízkoúrovňová knihovna pro práci s trojrozměrnou grafikou. OpenGL představuje jednotné API (Application Programming Interface) mezi programem a grafickým hardware. Standard API popisuje množinu funkcí a procedur s přesně specifikovaným chováním.

Zdrojový kód tohoto programu je v souboru `b2c_ogl.c`. Nejdříve jsou načteny všechny hlavičkové soubory, především `GL/glut.h` a `GL/glext.h`, což jsou právě knihovny OpenGL. Program podobně jako optimalizační program využívá seznam, jehož prvky reprezentují jednotlivé moduly v kontejneru. Který kontejner se má načíst ze souboru `bconfig.dat`, udává první parametr, se kterým je program spuštěn. Nejdříve jsou nastaveny vlastnosti okna a jsou registrovány všechny funkce, které zajišťují obsluhu událostí, jako například

tažení myši. Frekvence vykreslování je nastavena na 40-krát za sekundu. Poté je spuštěna nekonečná smyčka, která vykreslování zajišťuje. Ve funkci `onDisplay()` dojde k vykreslení kontejneru a všech modulů, které byly již "naloženy". Počet naložených modulů udává druhý parametr, se kterým byl program spuštěn. Vykreslování všech hran modulů či kontejneru je prováděno funkcí `glVertex3f()`.Ovládání celého programu stejně jako popis jsou uvedeny v sekci 4.4.2.

## 5.5 Testování

Testování probíhalo v průběhu celého cyklu implementace po jednotlivých částech tak, aby byly odstraněny všechny dílčí chyby. Po sestavení aplikace do jednoho celku proběhla druhá fáze testování, která měla potvrdit bezchybovost a správnost výpočtů.

Aplikace byla vyvíjena i testována na procesoru Intel Celeron, 2,4GHz. Během testování byla prováděna různá měření údajů, které klasifikují vlastnosti optimalizačního algoritmu. Některé naměřené výsledky jsou samozřejmě ovlivněny frekvencí procesoru a velikostí paměti, jako například doba výpočtu. Ostatní, jako je počet rekurzí či počet modulů umístěných v kontejneru, vlastnostmi procesoru ovlivněny nejsou. Vlastnosti algoritmu byly zkoumány na třech kategoriích testů.

První testovaní probíhalo na různých vstupních datech, tzn. s různými vstupními parametry či s různými vstupními soubory. Počet materiálů v aktuální den je vždy přibližně stejný, ovšem mění se jejich druh, tedy i druh modulů, které se mají naložit. Přehled výsledků je uveden v tabulce A.7. Z tabulky lze vyčíst, že naložení jednoho kontejneru z přibližně 3500 kusů modulů trvá přibližně stejnou dobu. Důvod je prostý, poměr množství skutečně naložených modulů od počtu modulů, ze kterých se vybírá, je velký. Tudíž je snadnější vybrat ty vhodné.

Zajímavější výsledky přineslo opakované měření se stejnými vstupními parametry a stejným vstupním souborem. Při naložení kontejneru se samozřejmě použité moduly ze seznamu odebrali a výpočet se opakoval s nižším počtem vstupních modulů. Počet rekurzí algoritmu v závislosti na počtu krabic je uveden v tabulce A.8. Doba výpočtu (tedy i počet rekurzí) se exponenciálně zvyšuje s klesajícím počtem modulů. V jisté úrovni již nelze naložení s danou procentuální mírou zaplnění vypočítat a výpočet je po 20 mil. rekurzí ukončen (viz. sekce 4.3.3).

V poslední kategorii testů jsem využil možnost optimalizačního programu potvrzovat vypočítaný výsledek. Tato vlastnost skýtá jednu z možností dalšího rozšíření aplikace (viz. kapitola 6.2). Optimalizační algoritmus provede výpočet, ale před uložením výsledku je uživatel dotázán, zda s výsledkem souhlasí. Pokud nesouhlasí, výpočet pokračuje dokud nedosáhne vyššího procenta zaplnění. Z takto prováděného testu zjistíme, jak je míra zaplnění závislá na době výpočtu (počtu rekurzí). Jak lze vyčíst z tabulky A.9, zvýšení zaplnění lze dosáhnout, jedná se však řádově o desetiny, maximálně jednotky procent, které nehrají ve výsledném naložení významnou roli. Algoritmus se sice snaží po odmítnutí vkládat do kontejneru jiné moduly, ovšem počet stejných modulů je značný, tudíž často dojde jen ke kosmetickým úpravám v naložení. To je důvod tak malého zlepšení. Jako vstupní data jsem použil různé dny ve stejném souboru, znázorněno též v tabulce.

### 5.5.1 Vlastnosti optimalizačního algoritmu

Testování do značné míry splnilo očekávání. Naměřené výsledky potvrdili teoretický předpoklad, že optimalizační algoritmus má exponenciální složitost v závislosti na počtu modulů.

Kritická hodnota, kdy se exponenciální křivka "láme", je někde mezi 1 000 a 1 500 moduly v závislosti na skladbě modulů. Dále je také patrné, že při vhodné skladbě modulů probíhá výpočet velmi rychle.

# Kapitola 6

# Výsledky a možnosti dalšího vývoje

V této poslední kapitole shrnu dosažené výsledky při vývoji, splnění požadavků a možnosti dalšího vývoje aplikace. Jelikož jsem aplikace ještě nemohl plně otestovat přímo v provozu jedná se o dílčí údaje a subjektivní názory odpovědných lidí, kteří mají zavádění aplikace na starosti.

## 6.1 Splnění požadavků

Po začlenění již zmiňovaného problému s přetěžováním modulů bude aplikace plně vyhovovat nárokům a splní všechny požadavky, které Škoda měla. Mám přislíbeno, že tento problém bude dořešen v nejbližších týdnech tak, aby celá aplikace byla po otestování spustitelná k 1. červenci 2007. Původní termín spuštění byl jaro 2007 s blíže nespecifikovaným datem. Toto se nepodařilo dodržet z důvodů jiných priorit v oddělení EOT (Aplikace pracovních procesů) společnosti Škoda, zejména pak zavádění systémů v nových montážních halách v Indii, což jsem nemohl ovlivnit.

Grafická rozhraní aplikace obsahují dokonce některé komponenty, které v původní specifikaci nebyly. Nejsou sice přímo nezbytné, ale usnadní práci zaměstnanců či zpřesní výpočet optimalizace. Po dohodě jsem proto doplnil možnost zadávat postačující míru zaplnění či zobrazení 3D síťového modelu naložení kontejneru.

Rychlost i způsob výpočtu jsou postačující k určenému použití. Vezmeme-li v úvahu, že se během jednoho dne balí a nakládají součástky do třech až čtyřech kontejnerů, nehraje pár sekund výpočtu naložení žádnou roli. Problém může nastat, pokud dojde k nějaké nečekané situaci. Například se do kontejneru na poslední chvíli musí vložit jiný materiál, než který byl v plánu. Tento problém ovšem musí a bude řešit vnitřní systém Škody, který nahrazený materiál zařadí zpět do cyklu výpočtu. Aplikace ze vstupních dat pouze vypočítá rozložení modulů v kontejneru a již se nestará, zda k naložení a vyexpedování skutečně došlo...

## 6.2 Možnosti dalšího vývoje

I když je aplikace již nyní plně využitelná, obsahuje velký potenciál rozšíření a využití. Jako vstupní data využívá informace ze souborů (viz. sekce 4.1.1). Pokud balící plán začne využívat jiný druh modulů pro balení součástek, nemusí se přeprogramovávat aplikace, ale stačí pouze změnit dané soubory. Proto lze využít aplikaci nejen ve skladech V8, kam byla

původně určena, ale i v jiných částech výroby, kde řeší podobný problém s nakládáním součástek.

Stěžejní část aplikace je samotný optimalizační program a na něj se chci také soustředit při dalším vývoji. V části 4.3.1 již byly naznačeny také možnosti jeho rozšíření. Program používá parametry, které nejsou pro aplikaci potřebné, avšak lze je dále využít. První dva parametry udávají počet dnů, které se mají vypočítat. Lze tedy vytvořit pole, kde pro každý prvek je vypočítáno samostatné naložení. Číslo kontejneru (šestý parametr) pak udává číslo prvního vypočítaného kontejneru, další mají číslo vždy o jedno vyšší. Dále lze využít čtvrtý parametr, který umožnuje dotazovat se uživatele, zda s vypočítaným výsledkem souhlasí. Této vlastnosti bylo využito i při testování optimalizačního algoritmu, viz. podkapitola 5.5. Mimo tyto možnosti je zde určitý prostor k zefektivnění výpočtu. A to například pokud bychom upravili průchod seznamem modulů či způsob skládání modulů do kontejneru.

Grafická rozhraní, ať již pro zadávání parametrů nebo pro zobrazování vypočítaných rozmístění modulů, sice lze také dále vyvíjet, upravovat rozmístění komponent či zlepšit zobrazování modulů v kontejneru. Nicméně jsem oba programy vytvořil tak, aby plně vyhovovali a plně postačovali způsobu jejich použití.

## 6.3 Závěr

Problém efektivní nakládky jakéhokoliv materiálu tíží většinu nejen expedičních firem. Aplikaci jsem proto vyvíjel především s ohledem na obecnost. Po menších úpravách, které se týkají hlavně druhu materiálu a stohovatelnosti modulů, věřím, že aplikace najde široké uplatnění v mnoha oblastech.

# Literatura


[1] J. Boyar. *The maximum resource bin packing problem.* Springer-Verlag, 2005.

[2] N. Samphaiboon. Heuristic and exact algorithms for the precedence-constrained knapsack problem. *Journal of Optimization Theory and Applications*, (3), 2000.

[3] B. Spell. *Java - Programujeme profesionálně.* Computer Press, 2002.

[4] Otevřená encyklopedie Wikipedia. Optimalizace. [online]. Dostupné na: http://cs.wikipedia.org/wiki/Optimalizace.

# Dodatek A

# Tabulky

| Modul | Pod modul |
|---|---|
| GLT5755/63 | pod GLT5754/56/62/65 |
| GLT54/56 | pod GLT5762/65 |
| GLT5764 | pro stohování pouze jeden typ palet |
| GLT5740/41 | pro stohování pouze tyto typ palet |
| GLT5744 | pro stohování pouze jeden typ palet |

Tabulka A.1: Stohování GLT modulů

| Modul | Počet kusů do GLT5754 | Počet kusů do GLT5755 |
|---|---|---|
| KLT001 | 8 | 4 |
| KLT002 | 16 | 8 |
| KLT003 | 18 | 8 |
| KLT004 | 36 | 16 |
| KLT005 | 84 | 40 |
| MOD009 | 4 | 1 |
| MOD010 | 8 | 2 |
| MOD011 | 8 | 4 |
| MOD012 | 8 | 3 |
| MOD013 | 24 | 9 |
| MOD014 | 16 | 6 |
| MOD015 | 18 | 6 |
| MOD016 | 36 | 12 |
| MOD017 | 36 | 12 |
| MOD018 | 36 | 12 |
| MOD019 | 18 | 4 |
| MOD020 | 36 | 9 |

Tabulka A.2: Seznam vkládání modulů KLT a MOD do modulů GLT5754 a GLT5755

| Modul | Šířka (rozměr x) | Hloubka (rozměr z) | Výška (rozměr y) | Nosnost (kg) |
|---|---|---|---|---|
| GLT000 | 1450 | 1120 | 585 | 300 |
| GLT001 | 1450 | 1120 | 695 | 300 |
| GLT002 | 2240 | 720 | 695 | 300 |
| GLT003 | 1096 | 688 | 695 | 300 |
| GLT004 | 1675 | 1500 | 715 | 300 |
| GLT5740 | 2000 | 1200 | 730 | 300 |
| GLT5741 | 1200 | 800 | 930 | 300 |
| GLT5744 | 1685 | 1285 | 1100 | 300 |
| GLT5754 | 1450 | 1130 | 730 | 300 |
| GLT5755 | 1130 | 825 | 730 | 300 |
| GLT5756 | 1450 | 1130 | 830 | 300 |
| GLT5762 | 2260 | 1450 | 730 | 300 |
| GLT5763 | 1130 | 725 | 370 | 300 |
| GLT5764 | 2260 | 1550 | 730 | 300 |
| GLT5765 | 2260 | 1450 | 1100 | 300 |
| KLT001 | 595 | 395 | 260 | 78 |
| KLT002 | 595 | 395 | 130 | 78 |
| KLT003 | 395 | 295 | 260 | 65 |
| KLT004 | 395 | 295 | 130 | 40 |
| KLT005 | 295 | 195 | 130 | 34 |
| MOD003 | 1040 | 688 | 550 | 122 |
| MOD004 | 1040 | 688 | 275 | 122 |
| MOD005 | 1040 | 688 | 183 | 122 |
| MOD006 | 1040 | 458 | 550 | 113 |
| MOD007 | 1040 | 458 | 275 | 113 |
| MOD008 | 1040 | 458 | 183 | 113 |
| MOD009 | 688 | 520 | 550 | 102 |
| MOD010 | 688 | 520 | 275 | 102 |
| MOD011 | 688 | 520 | 183 | 102 |
| MOD012 | 520 | 344 | 550 | 86 |
| MOD013 | 520 | 344 | 183 | 45 |
| MOD014 | 520 | 275 | 275 | 43 |
| MOD015 | 458 | 346 | 275 | 43 |
| MOD016 | 458 | 346 | 183 | 43 |
| MOD017 | 458 | 260 | 183 | 41 |
| MOD018 | 458 | 173 | 275 | 38 |
| MOD019 | 344 | 346 | 275 | 40 |
| MOD020 | 344 | 346 | 183 | 40 |
| MOD021 | 344 | 346 | 92 | 0 |
| MOD022 | 137 | 130 | 183 | 0 |
| MOD023 | 114 | 115 | 92 | 0 |

Tabulka A.3: Přehled nejpoužívanějších modulů

| Závod | Směna a den | Kód materiálu | Počet kusů | Závěska | Poznámky |
|---|---|---|---|---|---|
| 74 | 1NPO | 01M300032G | 7 | 1000003161 | 1 K |
| 74 | 1RPO | 02J300047M | 173 | 1000003177 | 1 K |
| 74 | 1OPO | 02J300048B | 15 | 1000003178 | 1 K |
| 74 | 1NPO | 038100037H | 29 | 1000003182 | 1 K |
| 74 | 1RUT | 038100054E | 1 | 1000003183 | 1 K |
| 74 | 1OUT | 038105264H | 173 | 1000003193 | 1 K |
| 74 | 1OUT | 06A100020MD | 3 | 1000003260 | 1 K |
| 74 | 1OST | 06A103724BD | 1 | 1000003261 | 1 K |
| 74 | 1NPA | 06A119518G | 1 | 1000003263 | 1 K |
| 74 | 1NPA | 06A133354G | 1 | 1000003265 | 1 K |
| 74 | 1RPA | 191611715 | 1 | 1000003291 | 1 K |
| 74 | 1OPA | 191823395 | 1 | 1000003292 | 1 K |
| 74 | 1RPA | 191971790A | 2 | 1000003296 | 1 K |
| 74 | 1OPA | 1C0906109A | 1 | 1000003299 | 1 K |
| 74 | 1RPA | 1C0909601 | 9 | 1000003300 | 1 K |
| 74 | 1NPA | 1H0611797 | 1 | 1000003307 | 1 K |
| 74 | 1RPA | 1H0819465E | 7 | 1000003308 | 1 K |
| 74 | 1OUT | 1H0867981 | 2 | 1000003312 | 1 K |
| 74 | 1RPA | 1H0877236 | 1 | 1000003313 | 1 K |
| 74 | 1OPA | 1H0919238 | 1 | 1000003316 | 1 K |

Tabulka A.4: Příklad obsahu souboru EXP_H_TETRIS.txt

| Závod | Materiál | Den | | Balení | Popis |
|---|---|---|---|---|---|
| 74 1 | 020110024A | (21.11.05) | 1 | GLT5763 | GLT 5763 (1130*725*370) |
| 74 1 | 020110024A | (21.11.05) | 2 | BSE040 | VCI–SÁČEK Z HADICE (400) |
| 74 1 | 020110024A | (21.11.05) | 3 | ZLG003 | PROLOŽKA GLT003 |
| 74 1 | 020110024A | (21.11.05) | 4 | GQG302 | HŘEBEN 4 PŘÍČNÝ |
| 74 1 | 020110024A | (21.11.05) | 5 | GLG201 | HŘEBEN 2 PODÉLNÝ |
| 74 1 | 02011024A | (8.2.2005) | 1 | GLT003 | GLT 3 (1096*688*695) |
| 74 1 | 02011024A | (8.2.2005) | 2 | DGL003 | VÍKO PRO GLT003 |
| 74 1 | 02011024A | (8.2.2005) | 3 | PGL003 | PALETA GLT3 |
| 74 1 | 02011024A | (8.2.2005) | 4 | WSR100 | VLNITÝ ROLOVANÝ PAPÍR (1000) |
| 74 1 | 02011024A | (8.2.2005) | 5 | BSE040 | VCI–SÁČEK Z HADICE (400) |
| 74 1 | 021117061B | (12.05.05) | 1 | KLT003 | KLT 3 (395*295*260) |
| 74 1 | 021117061B | (12.05.05) | 2 | BSN060 | PE–SÁČEK Z HADICE (600) |
| 74 1 | 02A911023L | (8.2.2005) | 1 | GLT003 | GLT 3 (1096*688*695) |
| 74 1 | 02A911023L | (8.2.2005) | 2 | DGL003 | VÍKO PRO GLT003 |
| 74 1 | 02A911023L | (8.2.2005) | 3 | PGL003 | PALETA GLT3 |
| 74 1 | 02A911023L | (8.2.2005) | 4 | WSR100 | VLNITÝ ROLOVANÝ PAPÍR (1000) |
| 74 1 | 02A911023L | (8.2.2005) | 5 | BSE040 | VCI-SÁČEK Z HADICE (400) |

Tabulka A.5: Příklad obsahu souboru EXP_THI_MODUL.txt

| Závěska | Umístění | | | Rotace | Velikost | | | Modul | Počet | Vložené |
|---|---|---|---|---|---|---|---|---|---|---|
| 1000004965 | 0 | 0 | 0 | 0 | 72 | 37 | 113 | GLT5763 | 1 | GLT5763 |
| 1000004907 | 72 | 0 | 0 | 0 | 72 | 37 | 113 | GLT5763 | 1 | GLT5763 |
| 1000006540 | 0 | 37 | 0 | 0 | 145 | 73 | 113 | GLT5754 | 18 | KLT003 |
| 1000004907 | 145 | 146 | 0 | 0 | 72 | 37 | 113 | GLT5763 | 1 | GLT5763 |
| 1000006104 | 145 | 0 | 0 | 0 | 82 | 73 | 113 | GLT5755 | 1 | GLT5755 |
| 1000004410 | 145 | 73 | 0 | 0 | 82 | 73 | 113 | GLT5755 | 1 | GLT5755 |
| 1000006540 | 0 | 147 | 0 | 0 | 145 | 73 | 113 | GLT5754 | 18 | KLT003 |
| 1000004907 | 145 | 183 | 0 | 0 | 72 | 37 | 113 | GLT5763 | 1 | GLT5763 |
| 1000004397 | 0 | 220 | 0 | 0 | 104 | 18 | 68 | MOD005 | 1 | MOD005 |
| 1000004395 | 145 | 220 | 0 | 0 | 68 | 18 | 104 | MOD005 | 1 | MOD005 |
| 1000004907 | 72 | 110 | 0 | 0 | 72 | 37 | 113 | GLT5763 | 1 | GLT5763 |
| 1000006210 | 104 | 220 | 0 | 0 | 39 | 13 | 59 | KLT002 | 1 | KLT002 |
| 1000006106 | 104 | 220 | 59 | 0 | 34 | 18 | 52 | MOD013 | 1 | MOD013 |
| 1000006019 | 0 | 220 | 68 | 0 | 104 | 18 | 45 | MOD008 | 1 | MOD008 |
| 1000006540 | 0 | 0 | 113 | 0 | 145 | 73 | 113 | GLT5754 | 18 | KLT003 |
| 1000006540 | 0 | 73 | 113 | 0 | 145 | 73 | 113 | GLT5754 | 18 | KLT003 |
| 1000006540 | 0 | 146 | 113 | 0 | 145 | 73 | 113 | GLT5754 | 18 | KLT003 |
| 1000006018 | 0 | 219 | 113 | 0 | 104 | 18 | 45 | MOD008 | 1 | MOD008 |
| 1000004119 | 104 | 219 | 113 | 0 | 39 | 13 | 59 | KLT002 | 1 | KLT002 |

Tabulka A.6: Příklad obsahu souboru res.dat

| Počet modulů na vstupu | Zaplnění [%] | Počet použitých modulů | Počet rekurzí |
|---|---|---|---|
| 3673 | 86,173 | 1153 | 3 736 |
| 2634 | 87,378 | 633 | 2 649 |
| 1122 | 86,400 | 166 | 499 |
| 2892 | 85,293 | 821 | 2 120 |
| 3112 | 87,482 | 1021 | 5 832 |
| 2923 | 86,932 | 931 | 3 121 |

Tabulka A.7: Test č. 1: Měření s různými vstupními daty

| Číslo výpočtu | Modulů na vstupu | Zaplnění [%] | Použitých modulů | Počer rekurzí |
|---|---|---|---|---|
| 1 | 3 673 | 86,173 | 1153 | 3 736 |
| 2 | 2 520 | 87,759 | 1439 | 3 086 |
| 3 | 1 081 | 85,021 | 539 | 219 483 |
| 4 | 542 | | | |
| 1 | 1 122 | 86,400 | 166 | 499 |
| 2 | 956 | 88,327 | 230 | 5618 |
| 3 | 726 | | | |
| 1 | 2 634 | 87,378 | 633 | 2 649 |
| 2 | 2 001 | 85,239 | 854 | 1 246 |
| 3 | 1 141 | 86,127 | 732 | 42 316 |
| 4 | 409 | | | |

Tabulka A.8: Test č. 2: Opakované měření se stejnými vstupními daty

| Číslo měření | Zaplnění [%] | Počet rekurzí |
|---:|---:|---:|
| 1 | 86,173 | 3 736 |
| 2 | 86,173 | 4 894 |
| 3 | 86,183 | 7 672 |
| 4 | 86,180 | 11 757 |
| 5 | 86,284 | 63 599 |
| 6 | 86,341 | 172 716 |
| 7 | 86,887 | 1 525 085 |
| | | - pondělí, 3673 modulů |
| 1 | 86,277 | 649 |
| 2 | 86,298 | 914 |
| 3 | 86,355 | 1 155 |
| 4 | 86,350 | 1 541 |
| 5 | 86,360 | 1 941 |
| 6 | 86,512 | 478 924 |
| 7 | 87,718 | 1 415 200 |
| | | - úterý, 1321 modulů |
| 1 | 86,400 | 499 |
| 2 | 86,421 | 551 |
| 3 | 86,441 | 565 |
| 4 | 86,441 | 747 |
| 5 | 86,437 | 239 459 |
| 6 | 86,341 | 172 716 |
| 7 | 87,537 | 4 108 760 |
| | | - středa, 1122 modulů |
| 1 | 87,121 | 1 726 |
| 2 | 87,317 | 2 495 |
| 3 | 87,507 | 14 366 |
| 4 | 87,532 | 15 143 |
| 5 | 87,886 | 308 875 |
| 6 | 87,899 | 5 227 108 |
| | | - čtvrtek, 1021 modulů |

Tabulka A.9: Test č. 3: Výpočty optimalizace při odmítání výsledku

# Dodatek B

# Uživatelský manuál

Dále bude následovat uživatelský manuál, který je napsán v MS Wordu, aby byl snadno aktualizovatelný dalšími pracovníky Škody. Má proto samostané číslování a mírně odlišný styl sazby...